Notes

Andrew Liu found out I said something wrong last week…

When overloading methods, function inputs must be different. Return types may be different

This changed in Java several versions ago!

```
String doubleMe(String s) { return s+s; }
double doubleMe(double val) { return 2.0*val; }
```

Last time…

We discussed

- Coding Style
- Testing

Java File I/O Revisited

```
File students = new File("/users/phaskell/CS112/roster.txt");

boolean students.canRead();
boolean students.canWrite();
boolean students.isDirectory();
boolean students.delete(); // careful!
boolean students.exists();
int students.length(); // if 'students' is a file
String[] students.list(); // if 'students' is a directory
```

A File just links to a file in the computer filesystem. Can't do any reading/writing with it

Java Stream Types

FileReader, FileWriter

- `FileWriter fw = new FileWriter(new File(path));`
- Reads (or writes) a char or an array of chars from the File
- Handles mapping the computer's native character set to (from) Java's UTF-16

PrintWriter

- Prints standard data types as text
- print(char), print(double), print(String), println(…), etc

(There are other Readers and Writers that handle bytes, objects, etc)

LOOK AT EXCERPT.java and run it!

Code-along

Read keyboard input until it ends

Convert to uppercase

Write to file “output.txt”

Open file and get length in bytes.

See if contains character ‘$’

Keyboard input ends if:

- Redirected from a file
- End-of-file character entered

Enumerations ("enums")

January, February, March, …

Red, Orange, Yellow, Green, …

Biology, Chemistry, Physics, Geology, …

Enumerated Types

Java allows you to define an *enumerated type*, which can then be used to declare variables

An enumerated type declaration lists all possible values for a variable of that type

The values are identifiers of your own choosing

The following declaration creates an enumerated type called `Season`

```
enum Season {winter, spring, summer, fall};
```

Any number of values can be listed

Enumerated Types

Once a type is defined, a variable of that type can be declared:

```
Season time;
```

And it can be assigned a value:

```
time = Season.fall;
```

The values are referenced through the name of the type

Enumerated types are *type-safe* – you cannot assign any value other than those listed

Ordinal Values

Internally, each value of an enumerated type is stored as an integer, called its *ordinal value*

The first value in an enumerated type has an ordinal value of zero, the second one, and so on

However, you cannot assign a numeric value to an enumerated type, even if it corresponds to a valid ordinal value

Enumerated Types

The declaration of an enumerated type is a special type of class, and each variable of that type is an object

The `ordinal` method returns the ordinal value of the object

The `name` method returns the name of the identifier corresponding to the object's value

See `IceCream.java`

```
//********************************************************************
//  IceCream.java       Author: Lewis/Loftus
//
//  Demonstrates the use of enumerated types.
//********************************************************************

public class IceCream
{
   enum Flavor {vanilla, chocolate, strawberry, fudgeRipple, coffee,
                rockyRoad, mintChocolateChip, cookieDough}

   //-----------------------------------------------------------------
   //  Creates and uses variables of the Flavor type.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Flavor cone1, cone2, cone3;

      cone1 = Flavor.rockyRoad;
      cone2 = Flavor.chocolate;

      System.out.println("cone1 value: " + cone1);
      System.out.println("cone1 ordinal: " + cone1.ordinal());
      System.out.println("cone1 name: " + cone1.name());
```

continued

continued

```
        System.out.println();
        System.out.println("cone2 value: " + cone2);
        System.out.println("cone2 ordinal: " + cone2.ordinal());
        System.out.println("cone2 name: " + cone2.name());

        cone3 = cone1;

        System.out.println();
        System.out.println("cone3 value: " + cone3);
        System.out.println("cone3 ordinal: " + cone3.ordinal());
        System.out.println("cone3 name: " + cone3.name());
    }
}
```

continued

```
        System.out.pri
        System.out.pri                                    );
        System.out.pri                                    e2.ordinal());
        System.out.pri                                    name());

        cone3 = cone1;

        System.out.pri
        System.out.pri                                    );
        System.out.pri                                    e3.ordinal());
        System.out.println("cone3 name: " + cone3.name());
    }
}
```

Output

```
cone1 value: rockyRoad
cone1 ordinal: 5
cone1 name: rockyRoad
cone2 value: chocolate
cone2 ordinal: 1
cone2 name: chocolate
cone3 value: rockyRoad
cone3 ordinal: 5
cone3 name: rockyRoad
```

Enumerations

For our class Card, could do an enumeration for the suit

Enumeration for the card values? Maybe, but since there are actual point values tied to the card values, we might want a full class. We want more functionality than we get from an enum.

Probably want enum in its own file, so it can be used by multiple other .java files:

- Card.java, CardDeck.java

CS112 –
Java
Programming

Fall 2022

Debugging

Debugging

I am among the world's greatest debuggers :)

Lessons:

- When there is a bug, at least one of your assumptions is wrong
- Divide the system into smaller pieces that are easier to understand
- Make theories but test them
- Keep separate lists of Things You Suspect vs Things You Verified from Testing
- "There is never just 1 bug"

Timestamps and buggy test equipment

- Processing time changed the timestamp time
- All test equipment was buggy!
- Graphs, eventually found “file mode”.

Debugging

So far I expect you have all been doing what is called "println debugging"

- Verify your assumptions by confirming values of key data at key points in the code
- Excellent and widely used strategy, but you have to keep on adding println() statements to your code, recompiling, and rerunning
- And sometimes bugs takes weeks to reproduce

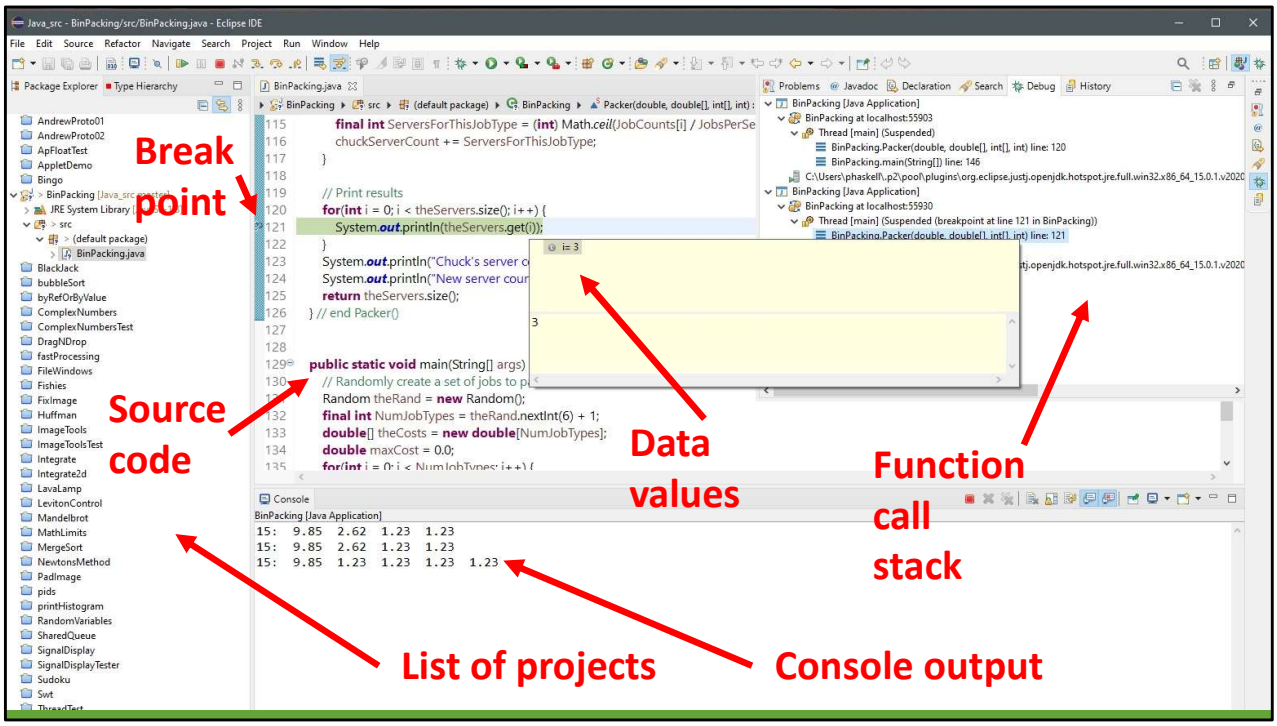
Look at buggy Plagiarism.java
Buggy but note great comments!

Graphical Debugger

A program that runs your program

- Can pause and later continue program at one or more points in code
- Can print out values at these "breakpoints" without having to add println's

For Java, many fine programs available. I use Eclipse, one of the most common


Break point
Source code
Data values
Function call stack
List of projects
Console output